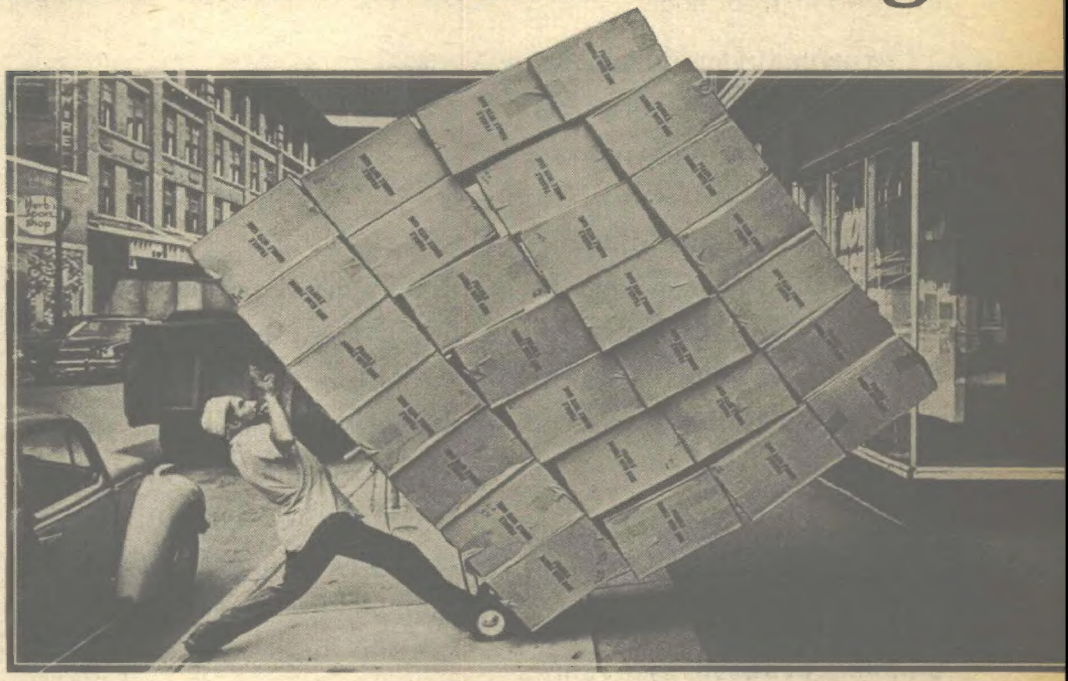
สุทธิศักดิ์ พงศ์ธนาพาณิช

เปิดโลกการเข้ารหัสข้อมูลด้วยภาษาซี

ตอน Huffman coding

ปฐมบทของการเสนอแนวคิดในการออกแบบโปรแกรมที่บีบอัดข้อมูลโดยวิธี Huffman



วิธีการเข้ารหัสข้อมูลที่มีประสิทธิภาพอีกวิธีที่รู้จักกันดีก็คือ Huffman coding ซึ่งได้ถูกคิดค้นโดย D.A. Huffman และได้ถูกตีพิมพ์ครั้งแรกในบทความ A Method for the Construction of Minimum Redundancy Codes ในปี ค.ศ. 1952 โดยส่วนใหญ่จะมีคุณสมบัติหลายอย่างที่เหมือนกับ Shannon-Fano coding คือสามารถสร้างโค้ดที่มีความยาว (จำนวนบิต) แตกต่างกันได้ ข้อมูลที่มีความน่าจะเป็นสูงกว่าก็จะถูกเข้ารหัสด้วยโค้ดที่มีจำนวนบิตน้อยกว่า และที่สำคัญโค้ดทุกตัวที่ได้จะมีแอตทริบิวต์ทำหน้าที่ไม่เหมือนกัน ดังนั้นโค้ดที่ได้จึงสามารถถอดรหัสได้อย่างถูกต้อง ส่วนขั้นตอนการถอดรหัสสายโค้ดโดยวิธี Huffman โดยส่วนใหญ่มักกระทำโดยใช้แผนภูมิต้นไม้ (binary decoder tree)

การสร้างแผนภูมิต้นไม้โดยวิธี Huffman นั้นมีอัลกอริทึมที่แตกต่างจากวิธี Shannon-Fano อย่างสิ้นเชิง เพราะแผนภูมิต้นไม้ของวิธี Shannon-Fano จะถูกสร้างจากบนลงล่าง ส่วนแผนภูมิต้นไม้ของวิธี Huffman จะถูกสร้างจากล่างขึ้นบน ดังนั้นโค้ดทุกตัวจะเริ่มถูกสร้างจากกิ่งก้านและไปสิ้นสุดที่ราก สำหรับโพรซีเดอร์ในการสร้างแผนภูมิต้นไม้ก็ค่อนข้างง่าย โดยเริ่มจากข้อมูลทุก ๆ ตัวจะเริ่มต้นที่โหนดกิ่งก้าน ซึ่งแต่ละโหนดจะมีค่าความน่าจะเป็นของแต่ละข้อมูลต่าง ๆ กัน ซึ่งขั้นตอนการสร้างสามารถเขียนได้ดังนี้

- ทำการนับค่าความน่าจะเป็นของข้อมูลทุกตัวและเรียงลำดับข้อมูล (sort) ตามค่าความน่าจะเป็นจากน้อยไปหามาก

- โหนดจะถูกสร้างทีละคู่โดยเริ่มจากคู่ข้อมูลที่มีค่าความน่าจะเป็นน้อยที่สุดไปหาคู่ข้อมูลที่มีค่าความน่าจะเป็นที่มากที่สุด แต่ถ้าหากข้อมูลตัวสุดท้ายไม่สามารถจัดเข้าคู่ได้คือเหลือเพียงตัวเดียวก็จะถูกแยกเป็นโหนดอิสระที่มีกิ่งเพียงกิ่งเดียว

- สร้างโหนดแม่ (parent node) ของโหนดคู่ที่ได้จากขั้นตอนแรก และจะมีค่าน้ำหนักเท่ากับผลรวมของค่าความน่าจะเป็นของโหนดลูก (child node) ทั้งสอง

- โหนดแม่ที่ได้ก็จะถูกเพิ่มเข้ามาเป็นโหนดอิสระ และโหนดลูกก็จะถูก

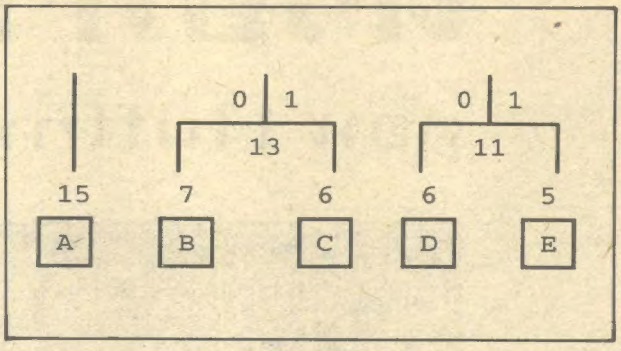
ยกเลิก

- โหนดลูกที่ถูกกำหนดให้เป็นเส้นทางของโหนดแม่โหนดหนึ่งจะถูกกำหนดให้มีค่าเป็นไบนารี่ 0 ส่วนโหนดลูกที่เหลือก็จะถูกกำหนดให้มีค่าเป็น 1
- ให้ทำซ้ำขั้นตอนทั้งหมดที่ผ่านมาจนกระทั่งเหลือโหนดอิสระเพียงโหนดเดียว ซึ่งโหนดที่เหลือนี้ก็จะถูกกำหนดเป็นโหนดราก (root node)

สำหรับกลุ่มข้อมูลที่ผู้เขียนใช้เป็นตัวอย่างแสดงการสร้างแผนภูมิต้นไม้โดยวิธี Huffman นี้ก็จะเหมือนกับบทความตอนที่ผ่านมา เพื่อให้ผู้อ่านสามารถเปรียบเทียบผลที่ได้ที่แตกต่างจากที่ได้จากวิธี Shannon-Fano โดยสมมติว่ากลุ่มข้อมูลประกอบด้วยตัวอักษร A ถึง E เมื่อนำมาเรียงลำดับตามค่าความถี่แล้วจะมีค่าความน่าจะเป็นดังรูปที่ 1

กลุ่มข้อมูล	A	B	C	D	E
ค่าความน่าจะเป็น	15	7	6	6	5

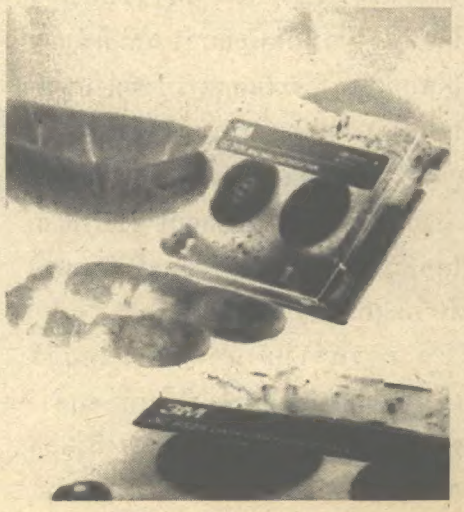
รูปที่ 1
กลุ่มข้อมูลเมื่อนำมาเรียงลำดับตามความถี่



รูปที่ 2
แสดงแผนภูมิต้นไม้ Huffman ขั้นตอนแรก

จากกลุ่มข้อมูลทั้ง 5 ตัวโดยเริ่มจากขั้นตอนแรกก็จะได้โหนดลูกทั้งหมด 5 โหนด และเนื่องจากจำนวนข้อมูลเป็นเลขคี่ ดังนั้นจึงสามารถจับคู่เพื่อสร้างโหนดคู่ได้ 2 โหนดและโหนดเดี่ยวอีก 1 โหนด โดยการจับคู่โหนดจะต้องเริ่มต้นจากคู่ที่มีค่าผลรวมของค่าความน่าจะเป็นน้อยที่สุดไปยังค่ามากที่สุด ดังนั้นโหนดคู่แรกจึงสามารถจับคู่ได้ระหว่าง D และ E (แต่ผู้อ่านสามารถที่จะเลือกจับคู่แรกเป็น C และ E ก็ได้ เพราะ C และ D มีค่าความถี่เท่ากัน ดังนั้นจึงไม่มีผลต่อค่าอัตราส่วนการบีบอัดข้อมูลที่ได้ ซึ่งมีน้ำหนักเท่ากับผลรวมระหว่าง 6 กับ 5 เป็น 11 และโหนดคู่ต่อมาก็จะเป็นการจับคู่ระหว่าง B และ C ซึ่งมีน้ำหนักเท่ากับผลรวมระหว่าง 7 กับ 6 เป็น 13 ส่วนตัวอักษร A ก็จะถูกปล่อยเป็นโหนดเดียวอิสระ ต่อจากนั้นก็ทำการสร้างโหนดแม่โดยเกิดจากผลรวมของโหนดลูกคู่ต่าง ๆ ดังนั้นโหนดแม่ของโหนดคู่ D/E ก็จะมีค่าน้ำหนักเป็น 11 และโหนดแม่ของโหนดคู่ B/C ก็จะมีค่าน้ำหนักเป็น 13 และต่อจากนั้นก็ทำการกำหนดค่าบิตให้กับแต่ละกิ่งก้าน ดังนั้นกิ่ง B และ D จึงมีค่าเป็นบิต 0 และกิ่ง C และ E ก็จะมีค่าบิตเป็น 1 ซึ่งแผนภูมิต้นไม้ที่ได้จากขั้นตอนแรกก็จะมีรูปร่างหน้าตาดังรูปที่ 2

ขั้นตอนต่อมาก็จัดการรวมโหนดคู่จากโหนดแม่ที่ได้ โดยสมมติว่าโหนดแม่ที่ได้ก็คือโหนดลูกของลำดับต่อไปนั่นเอง ดังนั้นโหนดแม่ของโหนดคู่ B/C และ D/E ก็จะถูกสร้างขึ้นมาโดยมีค่าน้ำหนักเท่ากับ 13+11 เท่ากับ 24 และกิ่งของโหนดแม่ B/C ก็จะถูกกำหนดให้มีค่าบิตเท่ากับ 0 ส่วนกิ่งของโหนดแม่ D/E ก็จะถูกกำหนดให้มีค่าบิตเท่ากับ 1 ส่วนโหนดเดียวอิสระ A นั้นก็จะถูกนำมาสร้างโหนดคู่ในขั้นตอนสุดท้ายของการสร้างแผนภูมิต้นไม้ เพื่อให้เหลือเพียงโหนดรากเพียงโหนดเดียว ดังนั้นโหนดกิ่ง A ก็จะถูกนำมาจับคู่กับโหนดที่ได้จากการรวมโหนดแม่ของโหนด B/C และ D/E ซึ่งโหนดรากที่ได้ก็จะมีค่าน้ำหนักเป็น 15+24 เท่ากับ 39 และเช่นเดิมกิ่งของโหนด A ก็จะถูกกำหนดให้มีค่าบิตเป็น 0 ส่วนโหนดที่ได้จากการจับคู่โหนดแม่ของ B/C และ D/E ก็จะถูกกำหนดให้มีค่าบิตเป็น 1 ซึ่งรูปที่ได้จะเป็นดังในรูปที่ 3

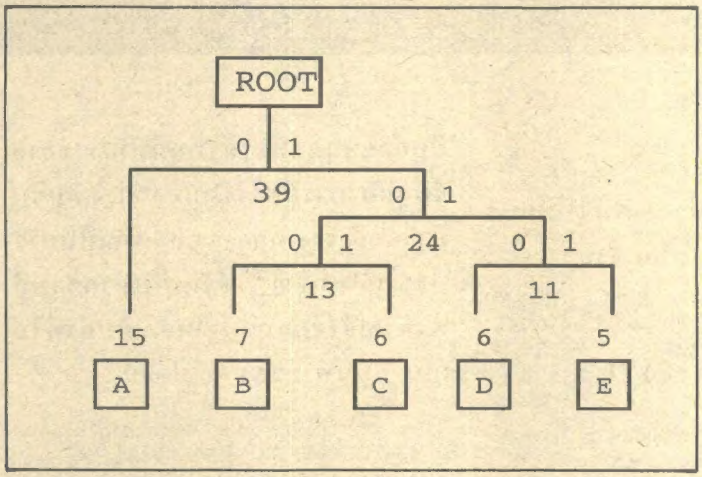


จากรูปที่ 3 เราสามารถที่จะเข้าหรือถอดรหัสสำหรับตัวอักษรแต่ละตัวได้โดยง่าย โดยการเดินตามกิ่งสาขาของต้นไม้ซึ่งจะได้โค้ดดังตัวอย่างในรูปที่ 4

จะเห็นได้ว่าข้อมูลที่มีค่าความถี่มากกว่าจะมีจำนวนบิตที่น้อยกว่า และเมื่อเปรียบเทียบกับเมื่อเราใช้ตัวอักษรรหัส ASCII ขนาด 8 บิตจะเห็นว่าข้อมูลทั้งหมดมีความยาวเท่ากับ 8×39 หรือ 312 บิต แต่โค้ดที่ได้จากวิธี Huffman จะทำให้ข้อมูลทั้งหมดมีความยาวเพียง 15+21+18+18+15 หรือ 87 บิตเท่านั้น ซึ่งสามารถสร้างโค้ดที่สั้นกว่าวิธี Shannon-Fano ซึ่งจะได้ข้อมูลที่มีความยาวทั้งหมด 89 บิต แต่โดยทั่วไปทั้งวิธี Huffman และ Shannon-Fano จะมีสมรรถนะที่ใกล้เคียงกันมาก โดยอย่างน้อยที่สุด Huffman จะมีประสิทธิภาพที่มากกว่าหรือเท่ากับ Shannon-Fano แต่ในการจัดเก็บข้อมูลโดยใช้โค้ดที่มีความยาวไม่เท่ากันนี้ไม่สามารถกระทำได้ในระบบคอมพิวเตอร์ ดังนั้นเราจึงนำโค้ดที่มีจำนวนบิตที่ไม่เท่ากันนี้มาเรียงต่อกันและนำมาเข้ารหัสกลับเป็นรหัส ASCII อีกครั้ง แต่เราจะต้องสร้างโพรซีเดอร์ที่สามารถถอดรหัสโค้ดเดิมที่มีความยาวไม่เท่ากันกลับคืนได้เช่นกัน

ออกแบบโปรแกรม Huffman

ต่อไปผมจะขออธิบายถึงขั้นตอนการออกแบบโปรแกรม Huffman โดยอาศัยวิธีการเลียนแบบขั้นตอนการสร้างแผนภูมิต้นไม้ของ Huffman จะเห็นว่าโหนดหนึ่ง ๆ ของต้นไม้จะประกอบด้วยข้อมูล 3 ตัวคือ ค่าน้ำหนัก และพอยน์เตอร์ไปยังโหนดลูกทั้งซ้ายและขวาโดยถูกกำหนดให้มีค่าบิตเป็น 0 และ 1 ตามลำดับ ดังนั้นเราจึงต้องกำหนดตัวแปรชนิดโครงสร้างสำหรับใช้เก็บ



รูปที่ 3
แสดงแผนภูมิต้นไม้ Huffman ที่สมบูรณ์

ตัวอักษร	A	B	C	D	E
โค็ด	0	100	101	110	111

ตัวอักษร	ค่าความถี่	Shannon-Fano Bit	รวม	Huffman Bit	รวม
A	15	2	30	1	15
B	7	2	14	3	21
C	6	2	12	3	18
D	6	3	18	3	18
E	5	3	15	3	15

รูปที่ 4 แสดงโค้ดที่ได้จากการถอดรหัส

ข้อมูลของแต่ละโหนดของต้นไม้เป็นดังนี้

```
typedef struct tree__node {
        unsiqned int count;
        int child__0;
        int child__1;
} NODE;
```

สำหรับรหัสแอสกีเราจะมีสัญลักษณ์อยู่ทั้งหมด 256 ตัว โดยมีค่าตั้งแต่ 0 ถึง 255 ดังนั้นเราจึงกำหนดให้ใช้ค่า 256 เป็นตัวชี้จุดสุดท้ายของสายข้อมูล ดังนั้นสัญลักษณ์ที่เราใช้ในการบีบอัดข้อมูลโดยวิธี Huffman มีทั้งหมด 257 ตัว ดังนั้นขนาดของต้นไม้ที่ใหญ่ที่สุดจึงมีขนาดเป็น 513 นั่นหมายความว่าเราจะต้องจองพื้นที่หน่วยความจำสำหรับตัวแปรอะเรย์ชนิดโครงสร้าง NODE ถึง 513 สมาชิก แต่สำหรับในโปรแกรมตัวอย่างนี้เราจะจองถึง 514 สมาชิกโดยกำหนดให้สมาชิกตัวสุดท้ายเป็นค่า dummy สำหรับการเปรียบเทียบในขณะที่กำลังสร้างแผนภูมิต้นไม้ และสำหรับการจัดเก็บโค้ดที่ได้จากการถอดรหัสจากแผนภูมิต้นไม้ ผมก็จะกำหนดให้โค้ดแต่ละตัวจัดเก็บลงในตัวแปรชนิดโครงสร้างดังนี้

```
typedef struct code {
        unsigned int code;
        int code__bits;
} CODE;
```

และเพื่อให้ผู้อ่านสามารถทำความเข้าใจถึงขั้นตอนการทำงานของโปรแกรมบีบอัดและขยายขนาดไฟล์โดยใช้วิธี Huffman นี้ก็ขอให้ดูจากขั้นตอนข้างล่างโดยที่รูทีน CompressFile จะทำหน้าที่บีบอัดไฟล์ตามที่กำหนดและรูทีน ExpandFile จะทำหน้าที่ขยายข้อมูลที่ถูกบีบอัดโดยรูทีนที่ผ่านมา

รูทีน CompressFile

1. จองเนื้อที่หน่วยความจำให้กับตัวแปรอะเรย์ ด้วยประโยคดังนี้

```
        conts = (unsigned long*) calloc
(256, sizeof(unsigned long));
        nodes = (NODE*) calloc (514, sizeof
(NODE));
        codes = (CODE*) calloc (257, sizeof
(CODE));
```

2. นับค่าความถี่ของข้อมูลทุก ๆ ตัวในไฟล์ที่ถูกกำหนด และจัดเก็บค่าความถี่ของข้อมูลแต่ละตัวลงในตัวแปรอะเรย์ counts[ ] ซึ่งถูกจัดการภายในรูทีน count__bytes

3. กำหนดขนาดของค่าความถี่ให้อยู่ในรูปแบบอัตราส่วนของค่าความถี่ที่มากที่สุด ซึ่งค่าความถี่ที่มากที่สุดนี้จะต้องถูกจัดเป็นจำนวนเท่าของ 255 เพื่อที่จะจำกัดขนาดของโค้ด Huffman ให้มีขนาดเพียง 16 บิต โดยที่ค่าความถี่ที่ถูกจัดการใหม่นี้จะถูกจัดเก็บลงในตัวแปร nodes[ ].count ซึ่งถูกจัดการภายในรูทีน scale__counts

4. จัดเก็บค่าความถี่ของข้อมูลแต่ละตัวที่ได้จากขั้นตอนที่ 3 ลงในไฟล์บีบอัดข้อมูล เพื่อให้รูทีนที่ทำการขยายข้อมูลสามารถสร้างโมเดลอย่างเดิมขึ้นมาได้อย่างถูกต้อง และเพื่อเป็นการประหยัดเนื้อที่ที่ใช้จัดเก็บค่าความถี่ทั้งหมดจึงขอแนะนำให้จัดเก็บค่าที่ไม่เป็น 0 เฉพาะช่วง ๆ ไปดังนี้

```
        start, stop, counts, start, stop, counts,
...,0
```

และท้ายสุดเราจะจัดเก็บค่า 0 เพื่อแสดงจุดสิ้นสุดของข้อมูลส่วนนี้ ซึ่งถูกจัดการภายในรูทีน output__counts

5. สร้างแผนภูมิต้นไม้ Huffman โดยเริ่มจากการค้นหา 2 โหนดที่มีค่าความถี่น้อยที่สุดแล้วบวกค่าความถี่ทั้งสองเข้าด้วยกันเพื่อกำหนดให้เป็นโหนดใหม่ ซึ่งโหนดใหม่ที่ได้นี้ก็จะมีโหนดลูก 2 โหนด ซึ่งโหนดทั้งสองก็จะจัดเก็บค่าความถี่ของโหนดที่น้อยที่สุดข้างต้น โดยจะจัดเก็บค่าความถี่ลงในตัวแปร nodes[ ].child__0 และ nodes[ ].child__1 ตามลำดับ หลังจากนั้นโหนดลูกทั้งสองก็จะถูกกำหนดสถานะให้ inactive แล้วก็เริ่มต้นประมวลผลซ้ำเดิมเพียงแต่มีการขยับโหนดที่

จะทำการประมวลผลถัดไปเรื่อย ๆ ซึ่งถูกจัดการภายในรูทีน build__tree

6. ทำการถอดรหัสโค้ด Huffman จากแผนภูมิต้นไม้ที่ถูกสร้างขึ้นมาในขั้นตอนที่ 5 โดยที่วิธีที่ง่ายที่สุดก็คือการใช้รูทีนแบบ recursive แล้วย้ายตัวเองลงมาตามแผนภูมิต้นไม้ทั้งซ้ายและขวาซึ่งถูกจัดการภายในรูทีน convert__tree__to__code

7. หลังจากนั้นจึงทำการบีบอัดข้อมูลโดยการแทนที่ข้อมูลแต่ละตัวด้วยโค้ด Huffman ที่ได้ แต่เนื่องจากโค้ด Huffman แต่ละตัวมีค่ายาว (จำนวนบิต) ไม่เท่ากัน แต่ในระบบการจัดเก็บข้อมูลสำหรับระบบไมโครคอมพิวเตอร์จะจัดเก็บโดยใช้โค้ดแอสกีซึ่งมีจำนวนบิตคงที่ ดังนั้นเราจึงต้องทำการนำโค้ด Huffman แต่ละตัวมาต่อกันและจัดเก็บลงในไฟล์เมื่อจำนวนบิตเท่ากับ 8 โดยในการจัดเก็บก็ยังคงในรหัสแอสกีอยู่เหมือนเดิม ซึ่งขั้นตอนการจัดเก็บข้อมูลนี้ได้ถูกจัดการในรูทีน OutputBits ซึ่งอยู่ในไฟล์ BITIO.C จากวารสารไมโครฉบับที่ผ่านมา และขั้นตอนที่ 7 นี้ได้ถูกเขียนลงในรูทีน compress__data ทั้งหมด

รูทีน ExpandFile

1. จองเนื้อที่หน่วยความจำให้กับตัวแปรอะเรย์ ด้วยประโยคดังนี้

```
nodes = (NODE*) calloc (514, sizeof (NODE));
```

2. ทำการย้อนกลับจากขั้นตอนที่ 4 ของรูทีน CompressFile โดยรูทีน input__counts

3. สร้างแผนภูมิต้นไม้ Huffman ตามขั้นตอนที่ 5 ของรูทีน CompressFile โดยรูทีน build__tree

4. ขยายขนาดข้อมูลทุกตัว ซึ่งขั้นตอนการขยายข้อมูลทำได้ยากกว่าการบีบอัดข้อมูล โดยเริ่มจากการอ่านข้อมูลจากไฟล์แล้วประมวลผลครั้งละ 1 บิต ซึ่งขั้นตอนนี้ได้ถูกจัดการภายในรูทีน InputBit โดยเริ่มต้นจากรากของแผนภูมิต้นไม้แล้วเดินตามกิ่งก้าน child__0 หรือ child__1 ทั้งนี้ก็ขึ้นกลับแต่ละบิตว่ามีค่าเป็น 0 หรือ 1 ตามลำดับ ซึ่งจะได้โค้ดแต่ละตัวก็ต่อเมื่อเราอ่านบิตแล้วเคลื่อนลงมาจนถึงโหนดล่างสุดของต้นไม้ หลังจากนั้นจึงจัดเก็บรหัสแอสกีลงในไฟล์ที่ขยายขนาด ซึ่งการขยายข้อมูลจะสิ้นสุดเมื่อข้อมูลที่อ่านได้ค่าที่อ่านได้มีค่าเท่ากับ END__OF__STREAM ซึ่งเป็นค่าคงที่ที่ถูกกำหนดให้มีค่าเท่ากับ 256 โดยขั้นตอนทั้งหมดอยู่ภายในรูทีน expand__data

จากเนื้อหาข้างต้นทั้งหมดผู้อ่านคงสามารถได้แนวคิดในการออกแบบโปรแกรมที่บีบอัดข้อมูลโดยวิธี Huffman ได้ ส่วนลูกเล่นต่าง ๆ ก็สามารถเพิ่มได้ตามอัธยาศัยครับ ส่วนในฉบับต่อไปก็จะแสดงโปรแกรมที่บีบอัดและขยายข้อมูลโดยวิธี Huffman รวมทั้งโปรแกรมใช้งานรูทีน CompressFile และ ExpandFile ติดตามกันต่อไปครับ

หนังสือดี ซีเอ็ด

การพัฒนาระบบฐานข้อมูลด้วยภาษาซี

■ *โดย อ็อล สตีเวนส์*
เรียบเรียงโดย อนุชิต สุมธิวิทย์/วีร์ ตั้งมั่นภักดีพงษ์
ราคาเล่มละ 150 บาท

หนังสือเล่มนี้ประกอบด้วยพื้นฐานและการออกแบบข้อมูลเบื้องต้น วิธีการสร้างระบบจัดการฐานข้อมูลอย่างง่ายด้วยภาษาซี พร้อมโปรแกรมที่สามารถนำไปใช้งานได้จริง ผู้อ่านจึงควรมีความสามารถในการเขียนโปรแกรมด้วยภาษาซีอยู่บ้างแล้ว นอกจากนี้จุดประสงค์ของหนังสือต้องการให้ผู้อ่านสามารถสร้าง tool เพื่อลดเวลาในการโปรแกรมลงได้อีกด้วย หนังสือเล่มนี้จึงเหมาะสำหรับผู้ที่สนใจเรื่องฐานข้อมูล และผู้ที่รักการเขียนโปรแกรมทั่วไป

หาซื้อได้ตามร้านหนังสือทั่วไป
หรือสั่งซื้อโดยตรงได้ที่

บริษัท ซีเอ็ดยูเคชั่น จำกัด
800/43-45 ซอยตระกูลสุข ถนนอโศก-ดินแดง ห้วยขวาง กรุงเทพฯ 10400 โทร. 248-6280-9 (เวลาทำการ จันทร์-ศุกร์ 8.10-17.30 น.) โดยส่งธนาณัติสั่งจ่าย ปท.ดินแดง ในนามบริษัทฯ หรือเช็คส่วนตัวของธนาคารในเขตกรุงเทพฯ พร้อมค่าจัดส่ง
ค่าจัดส่ง สั่งซื้อไม่เกิน 100 บาท ค่าจัดส่ง 8 บาท สั่งซื้อเกิน 100 บาท ค่าจัดส่ง 8%
พิเศษ สำหรับสมาชิกวารสารของซีเอ็ด สั่งซื้อโดยตรงที่บริษัทฯ ได้ในราคาลด 15% เพิ่มค่าลงทะเบียน 4 บาท ค่าจัดส่งฟรี โดยระบุหมายเลขสมาชิกด้วย